Hexo 中文文档

wizardforcel

Published
with GitBook



目錄

Hexo 中文文档

来源：Hexo

```
$ npm install hexo-cli -g
$ hexo init blog
$ cd blog
$ npm install
$ hexo server
```

文档

开始使用

概述

欢迎使用 Hexo，本文档将帮助您快速上手。如果您在使用过程中遇到问题，请查看 问题解答 中的解答，或者在 GitHub、Google Group 上提问。

什么是 Hexo？

Hexo 是一个快速、简洁且高效的博客框架。Hexo 使用 Markdown（或其他渲染引擎）解析文章，在几秒内，即可利用靓丽的主题生成静态网页。

安装

安装 Hexo 只需几分钟时间，若您在安装过程中遇到问题或无法找到解决方式，请 提交问题，我会尽力解决您的问题。

安装前提

安装 Hexo 相当简单。然而在安装前，您必须检查电脑中是否已安装下列应用程序：

- Node.js
- Git

如果您的电脑中已经安装上述必备程序，那么恭喜您！接下来只需要使用 npm 即可完成 Hexo 的安装。

```
$ npm install -g hexo-cli
```

如果您的电脑中尚未安装所需要的程序，请根据以下安装指示完成安装。

> **Mac** 用户 > > 您在编译时可能会遇到问题，请先到 App Store 安装 Xcode，Xcode 完成后，启动并进入 **Preferences -> Download -> Command Line Tools -> Install** 安装命令行工具。

安装 Git

- Windows：下载并安装 git.
- Mac：使用 Homebrew, MacPorts 或下载 安装程序 安装。
- Linux (Ubuntu, Debian)：`sudo apt-get install git-core`
- Linux (Fedora, Red Hat, CentOS)：`sudo yum install git-core`

安装 Node.js

安装 Node.js 的最佳方式是使用 nvm。

cURL:

```
$ curl https://raw.github.com/creationix/nvm/master/install.sh | sh
```

Wget:

```
$ wget -qO- https://raw.github.com/creationix/nvm/master/install.sh
```

安装完成后，重启终端并执行下列命令即可安装 Node.js。

```
$ nvm install 4
```

或者您也可以下载 安装程序 来安装。

安装 Hexo

所有必备的应用程序安装完成后，即可使用 npm 安装 Hexo。

```
$ npm install -g hexo-cli
```

建站

安装 Hexo 完成后，请执行下列命令，Hexo 将会在指定文件夹中新建所需要的文件。

```
$ hexo init <folder>$ cd
$ npm install
```

新建完成后，指定文件夹的目录如下：

```
.
├── _config.yml
├── package.json
├── scaffolds
├── source
|   ├── _drafts
|   └── _posts
└── themes
```

_config.yml

网站的 配置 信息，您可以在此配置大部分的参数。

package.json

应用程序的信息。EJS, Stylus 和 Markdown renderer 已默认安装，您可以自由移除。

```
{  "name"hexo-site",  "version"0.0.0",  "privatetrue,  "hexo{    "v
  ,  "dependencies{    "hexo"^3.0.0",    "hexo-generator-archive"^(
```

scaffolds

模版 文件夹。当您新建文章时，Hexo 会根据 scaffold 来建立文件。

source

资源文件夹是存放用户资源的地方。除 `_posts` 文件夹之外，开头命名为 `_` (下划线)的文件 / 文件夹和隐藏的文件将会被忽略。Markdown 和 HTML 文件会被解析并放到 `public` 文件夹，而其他文件会被拷贝过去。

themes

主题 文件夹。Hexo 会根据主题来生成静态页面。

配置

您可以在 `_config.yml` 中修改大部份的配置。

网站

参数	描述
`title`	网站标题
`subtitle`	网站副标题
`description`	网站描述
`author`	您的名字
`language`	网站使用的语言
`timezone`	网站时区。Hexo 默认使用您电脑的时区。时区列表。比如说： `America/New_York` , `Japan` , 和 `UTC` 。

网址

参数	描述	默认值
`url`	网址	
`root`	网站根目录	
`permalink`	文章的 永久链接 格式	`:year/:month/:day/:title/`
`permalink_default`	永久链接中各部分的默认值	

> 网站存放在子目录 > > 如果您的网站存放在子目录中，例如 `http://yoursite.com/blog` ，则请将您的 `url` 设为 `http://yoursite.com/blog` 并把 `root` 设为 `/blog/` 。

目录

参数	描述	默认值
`source_dir`	资源文件夹，这个文件夹用来存放内容。	`source`
`public_dir`	公共文件夹，这个文件夹用于存放生成的站点文件。	`public`
`tag_dir`	标签文件夹	`tags`
`archive_dir`	归档文件夹	`archives`
`category_dir`	分类文件夹	`categories`
`code_dir`	Include code 文件夹	`downloads/code`
`i18n_dir`	国际化（i18n）文件夹	`:lang`
`skip_render`	跳过指定文件的渲染，您可使用 glob 表达式来匹配路径。	

文章

参数	描述	默认值
`new_post_name`	新文章的文件名称	:title.md
`default_layout`	预设布局	post
`auto_spacing`	在中文和英文之间加入空格	false
`titlecase`	把标题转换为 title case	false
`external_link`	在新标签中打开链接	true
`filename_case`	把文件名称转换为 (1) 小写或 (2) 大写	0
`render_drafts`	显示草稿	false
`post_asset_folder`	启动 Asset 文件夹	false
`relative_link`	把链接改为与根目录的相对位址	false
`future`	显示未来的文章	true
`highlight`	代码块的设置	

分类 & 标签

参数	描述	默认值
`default_category`	默认分类	`uncategorized`
`category_map`	分类别名	
`tag_map`	标签别名	

日期 / 时间格式

Hexo 使用 Moment.js 来解析和显示时间。

参数	描述	默认值
`date_format`	日期格式	`MMM D YYYY`
`time_format`	时间格式	`H:mm:ss`

分页

参数	描述	默认值
`per_page`	每页显示的文章量 (0 = 关闭分页功能)	`10`
`pagination_dir`	分页目录	`page`

扩展

参数	描述
`theme`	当前主题名称。值为 `false` 时禁用主题
`deploy`	部署部分的设置

指令

init

```
$ hexo init [folder]
```

新建一个网站。如果没有设置 `folder` ，Hexo 默认在目前的文件夹建立网站。

new

```
$ hexo new [layout] <title>
```

新建一篇文章。如果没有设置 `layout` 的话，默认使用 _config.yml 中的 `default_layout` 参数代替。如果标题包含空格的话，请使用引号括起来。

generate

```
$ hexo generate
```

生成静态文件。

选项	描述
`-d` , `--deploy`	文件生成后立即部署网站
`-w` , `--watch`	监视文件变动

publish

```
$ hexo publish [layout] <filename>
```

发表草稿。

server

```
$ hexo server
```

启动服务器。默认情况下，访问网址为： `http://localhost:4000/` 。

选项	描述
`-p` , `--port`	重设端口
`-s` , `--static`	只使用静态文件
`-l` , `--log`	启动日记记录，使用覆盖记录格式

deploy

```
$ hexo deploy
```

部署网站。

参数	描述
`-g` , `--generate`	部署之前预先生成静态文件

render

```
$ hexo render <file1> [file2] ...
```

渲染文件。

参数	描述
`-o` , `--output`	设置输出路径

migrate

```
$ hexo migrate <type
```

从其他博客系统 迁移内容。

clean

```
$ hexo clean
```

清除缓存文件 (`db.json`) 和已生成的静态文件 (`public`)。

list

```
$ hexo list <type
```

列出网站资料。

version

```
$ hexo version
```

显示 Hexo 版本。

选项

安全模式

```
$ hexo --safe
```

在安全模式下，不会载入插件和脚本。当您在安装新插件遭遇问题时，可以尝试以安全模式重新执行。

调试模式

```
$ hexo --debug
```

在终端中显示调试信息并记录到 `debug.log` 。当您碰到问题时，可以尝试用调试模式重新执行一次，并 提交调试信息到 GitHub。

简洁模式

```
$ hexo --silent
```

隐藏终端信息。

自定义配置文件的路径

```
$ hexo --config custom.yml
```

自定义配置文件的路径，执行后将不再使用 `_config.yml` 。

显示草稿

```
$ hexo --draft
```

显示 `source/_drafts` 文件夹中的草稿文章。

自定义 CWD

```
$ hexo --cwd /path/to/cwd
```

自定义当前工作目录（Current working directory）的路径。

迁移

RSS

首先，安装 `hexo-migrator-rss` 插件。

```
$ npm install hexo-migrator-rss --save
```

插件安装完成后，执行下列命令，从 RSS 迁移所有文章。 `source` 可以是文件路径或网址。

```
$ hexo migrate rss <source
```

Jekyll

把 `_posts` 文件夹内的所有文件复制到 `source/_posts` 文件夹，并在 `_config.yml` 中修改 `new_post_name` 参数。

```
new_post_name: :year-:month-:day-:title.md
```

Octopress

把 Octopress `source/_posts` 文件夹内的所有文件转移到 Hexo 的 `source/_posts` 文件夹，并修改 `_config.yml` 中的 `new_post_name` 参数。

```
new_post_name: :year-:month-:day-:title.md
```

WordPress

首先，安装 `hexo-migrator-wordpress` 插件。

```
$ npm install hexo-migrator-wordpress --save
```

在 WordPress 仪表盘中导出数据(“Tools” → “Export” → “WordPress”)（详情参考 WP支持页面）。

插件安装完成后，执行下列命令来迁移所有文章。 `source` 可以是 WordPress 导出的文件路径或网址。

```
$ hexo migrate wordpress <source
```

Joomla

首先，安装 `hexo-migrator-joomla` 插件。

```
$ npm install hexo-migrator-joomla --save
```

使用 J2XML 组件导出 Joomla 文章。 插件安装完成后，执行下列命令来迁移所有文章。 `source` 可以是 Joomla 导出的文件路径或网址。

```
$ hexo migrate joomla <source
```

基本操作

写作

你可以执行下列命令来创建一篇新文章。

```
$ hexo new [layout] <title>
```

您可以在命令中指定文章的布局（layout），默认为 `post`，可以通过修改 `_config.yml` 中的 `default_layout` 参数来指定默认布局。

布局（Layout）

Hexo 有三种默认布局：`post`、`page` 和 `draft`，它们分别对应不同的路径，而您自定义的其他布局和 `post` 相同，都将储存到 `source/_posts` 文件夹。

布局	路径
`post`	`source/_posts`
`page`	`source`
`draft`	`source/_drafts`

> 不要处理我的文章 > > 如果你不想你的文章被处理，你可以将 Front-Matter 中的 `layout:` 设为 `false` 。

文件名称

Hexo 默认以标题做为文件名称，但您可编辑 `new_post_name` 参数来改变默认的文件名称，举例来说，设为 `:year-:month-:day-:title.md` 可让您更方便的通过日期来管理文章。

变量	描述
`:title`	标题（小写，空格将会被替换为短杠）
`:year`	建立的年份，比如， `2015`
`:month`	建立的月份（有前导零），比如， `04`
`:i_month`	建立的月份（无前导零），比如， `4`
`:day`	建立的日期（有前导零），比如， `07`
`:i_day`	建立的日期（无前导零），比如， `7`

草稿

刚刚提到了 Hexo 的一种特殊布局：`draft`，这种布局在建立时会被保存到 `source/_drafts` 文件夹，您可通过 `publish` 命令将草稿移动到 `source/_posts` 文件夹，该命令的使用方式与 `new` 十分类似，您也可在命令中指定 `layout` 来指定布局。

```
$ hexo publish [layout] <title>
```

草稿默认不会显示在页面中，您可在执行时加上 `--draft` 参数，或是把 `render_drafts` 参数设为 `true` 来预览草稿。

模版（Scaffold）

在新建文章时，Hexo 会根据 `scaffolds` 文件夹内相对应的文件来建立文件，例如：

```
$ hexo new photo "My Gallery"
```

在执行这行指令时，Hexo 会尝试在 `scaffolds` 文件夹中寻找 `photo.md`，并根据其内容建立文章，以下是您可以在模版中使用的变量：

变量	描述
`layout`	布局
`title`	标题
`date`	文件建立日期

Front-matter

Front-matter 是文件最上方以 `---` 分隔的区域，用于指定个别文件的变量，举例来说：

```
title: Hello World
date: 2013/7/13 20:46:25
---
```

以下是预先定义的参数，您可在模板中使用这些参数值并加以利用。

参数	描述	默认值
`layout`	布局	
`title`	标题	
`date`	建立日期	文件建立日期
`updated`	更新日期	文件更新日期
`comments`	开启文章的评论功能	true
`tags`	标签（不适用于分页）	
`categories`	分类（不适用于分页）	
`permalink`	覆盖文章网址	

分类和标签

只有文章支持分类和标签，您可以在 Front-matter 中设置。在其他系统中，分类和标签听起来很接近，但是在 Hexo 中两者有着明显的差别：分类具有顺序性和层次性，也就是说 `Foo, Bar` 不等于 `Bar, Foo`；而标签没有顺序和层次。

```
categories:
- Diary
tags:
- PS3
- Games
```

JSON Front-matter

除了 YAML 外，你也可以使用 JSON 来编写 Front-matter，只要将 `---` 代换成 `;;;` 即可。

```
"title": "Hello World","date": "2013/7/13 20:46:25";;;
```

标签插件（Tag Plugins）

标签插件和 Front-matter 中的标签不同，它们是用于在文章中快速插入特定内容的插件。

引用块

在文章中插入引言，可包含作者、来源和标题。

别号： quote

```
{% blockquote [author[, source]] [link] [source_link_title] %}
content
{% endblockquote %}
```

样例

没有提供参数，则只输出普通的 **blockquote**

```
{% blockquote %}
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesc
{% endblockquote %}
```

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque hendrerit lacus ut purus iaculis feugiat. Sed nec tempor elit, quis aliquam neque. Curabitur sed diam eget dolor fermentum semper at eu lorem.

引用书上的句子

```
{% blockquote David Levithan, Wide Awake %}
Do not just seek happiness for yourself. Seek happiness for all. Th
{% endblockquote %}
```

> Do not just seek happiness for yourself. Seek happiness for all. Through kindness. Through mercy. > > <footer>**David Levithan**<cite>Wide Awake</cite></footer>

引用 **Twitter**

```
{% blockquote @DevDocs https://twitter.com/devdocs/status/35609519
NEW: DevDocs now comes with syntax highlighting. http://devdocs.io
{% endblockquote %}
```

> NEW: DevDocs now comes with syntax highlighting. http://devdocs.io > > <footer>**@DevDocs**<cite>twitter.com/devdocs/status/356095192085962752</cite></footer>

引用网络上的文章

```
{% blockquote Seth Godin http://sethgodin.typepad.com/seths_blog/2
Every interaction is both precious and an opportunity to delight.
{% endblockquote %}
```

> Every interaction is both precious and an opportunity to delight. > > <footer>**Seth Godin**<cite>Welcome to Island Marketing</cite></footer>

代码块

在文章中插入代码。

别名： code

```
{% codeblock [title] [lang:language] [url] [link text] %}
code snippet
{% endcodeblock %}
```

样例

普通的代码块

```
{% codeblock %}
alert('Hello World!');
{% endcodeblock %}
```

```
alert('Hello World!');
```

指定语言

```
{% codeblock lang:objc %}
[rectangle setX: 10 y: 10 width: 20 height: 20];
{% endcodeblock %}
```

```
[rectangle setX: 10102020
```

附加说明

```
{% codeblock Array.map %}
array.map(callback[, thisArg])
{% endcodeblock %}
```

```
array.map(callback[, thisArg])
```

附加说明和网址

```
{% codeblock _.compact http://underscorejs.org/#compact Underscore
_.compact([0, 1, false, 2, '', 3]);
=> [1, 2, 3]
{% endcodeblock %}
```

Underscore.js

```
_.compact([0, 1, false, 2, '', 3]);
=> [1, 2, 3]
```

反引号代码块

另一种形式的代码块，不同的是它使用三个反引号来包裹。

`[language] [title] [url] [link text] code snippet`

Pull Quote

在文章中插入 Pull quote。

```
{% pullquote [class] %}
content
{% endpullquote %}
```

jsFiddle

在文章中嵌入 jsFiddle。

```
{% jsfiddle shorttag [tabs] [skin] [width] [height] %}
```

Gist

在文章中嵌入 Gist。

```
{% gist gist_id [filename] %}
```

iframe

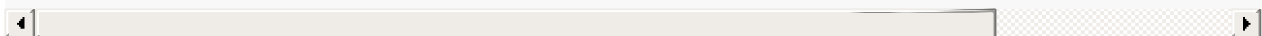
在文章中插入 iframe。

```
{% iframe url [width] [height] %}
```

Image

在文章中插入指定大小的图片。

```
{% img [class names] /path/to/image [width] [height] [title text [a
```



Link

在文章中插入链接，并自动给外部链接添加 `target="_blank"` 属性。

```
{% link text url [external] [title] %}
```

Include Code

插入 `source` 文件夹内的代码文件。

```
{% include_code [title] [lang:language] path/to/file %}
```

Youtube

在文章中插入 Youtube 视频。

```
{% youtube video_id %}
```

Vimeo

在文章中插入 Vimeo 视频。

```
{% vimeo video_id %}
```

引用文章

引用其他文章的链接。

```
{% post_path slug %}
{% post_link slug [title] %}
```

引用资源

引用文章的资源。

```
{% asset_path slug %}
{% asset_img slug [title] %}
{% asset_link slug [title] %}
```

Raw

如果您想在文章中插入 Swig 标签，可以尝试使用 Raw 标签，以免发生解析异常。

```
{% raw %}
content
{% endraw %}
```

资源文件夹

资源（Asset）代表 `source` 文件夹中除了文章以外的所有文件，例如图片、CSS、JS 文件等。比方说，如果你的Hexo项目中只有少量图片，那最简单的方法就是将它们放在 `source/images` 文件夹中。然后通过类似于 `![](/images/image.jpg)` 的方法访问它们。

文章资源文件夹

对于那些想要更有规律地提供图片和其他资源以及想要将他们的资源分布在各个文章上的人来说，Hexo也提供了更组织化的方式来管理资源。这个稍微有些复杂但是管理资源非常方便的功能可以通过将 `config.yml` 文件中的 `post_asset_folder` 选项设为 `true` 来打开。

```
post_asset_folder: true
```

当资源文件管理功能打开后，Hexo将会在你每一次通过 `hexo new [layout] <title>` 命令创建新文章时自动创建一个文件夹。这个资源文件夹将会有与这个 markdown 文件一样的名字。将所有与你的文章有关的资源放在这个关联文件夹中之后，你可以通过相对路径来引用它们，这样你就得到了一个更简单而且方便得多的工作流。

相对路径引用的标签插件

通过常规的 markdown 语法和相对路径来引用图片和其它资源可能会导致它们在存档页或者主页上显示不正确。在Hexo 2时代，社区创建了很多插件来解决这个问题。但是，随着Hexo 3 的发布，许多新的标签插件被加入到了核心代码中。这使得你可以更简单地在文章中引用你的资源。

```
{% asset_path slug %}
{% asset_img slug [title] %}
{% asset_link slug [title] %}
```

比如说：当你打开文章资源文件夹功能后，你把一个 `example.jpg` 图片放在了你的资源文件夹中，如果通过使用相对路径的常规 markdown 语法 `![](/example.jpg)` ，它将 不会 出现在首页上。（但是它会在文章中按你期待的方式工作）

正确的引用图片方式是使用下列的标签插件而不是 markdown ：

```
{% asset_img example.jpg This is an example image %}
```

通过这种方式，图片将会同时出现在文章和主页以及归档页中。

数据文件

有时您可能需要在主题中使用某些资料，而这些资料并不在文章内，并且是需要重复使用的，那么您可以考虑使用 Hexo 3.0 新增的「数据文件」功能。此功能会载入 `source/_data` 内的 YAML 或 JSON 文件，如此一来您便能在网站中复用这些文件了。

举例来说，在 `source/_data` 文件夹中新建 `menu.yml` 文件：

```
Home: /
Gallery: /gallery/
Archives: /archives/
```

您就能在模板中使用这些资料：

```
{% for link in site.data.menu %}
  <a href="{{ link }}">{{ loop.key }}</a>
{% endfor %}
```

服务器

hexo-server">hexo-server

Hexo 3.0 把服务器独立成了个别模块，您必须先安装 hexo-server 才能使用。

```
$ npm install hexo-server --save
```

安装完成后，输入以下命令以启动服务器，您的网站会在 `http://localhost:4000` 下启动。在服务器启动期间，Hexo 会监视文件变动并自动更新，您无须重启服务器。

```
$ hexo server
```

如果您想要更改端口，或是在执行时遇到了 `EADDRINUSE` 错误，可以在执行时使用 `-p` 选项指定其他端口，如下：

```
$ hexo server -p 5000
```

静态模式

在静态模式下，服务器只处理 `public` 文件夹内的文件，而不会处理文件变动，在执行时，您应该先自行执行 `hexo generate`，此模式通常用于生产环境（production mode）下。

```
$ hexo server -s
```

自定义 IP

服务器默认运行在 `0.0.0.0`，您可以覆盖默认的 IP 设置，如下：

```
$ hexo server -i 192.1681.1
```

Pow

Pow 是一个 Mac 系统上的零配置 Rack 服务器，它也可以作为一个简单易用的静态文件服务器来使用。

安装

```
$ curl get.pow.cx | sh
```

设置

在 `~/.pow` 文件夹建立链接（symlink）。

```
$ cd
$ ln -s
```

您的网站将会在 `http://myapp.dev` 下运行，网址根据链接名称而定。

生成文件

使用 Hexo 生成静态文件快速而且简单。

```
$ hexo generate
```

监视文件变动

Hexo 能够监视文件变动并立即重新生成静态文件，在生成时会比对文件的 SHA1 checksum，只有变动的文件才会写入。

```
$ hexo generate --watch
```

完成后部署

您可执行下列的其中一个命令，让 Hexo 在生成完毕后自动部署网站，两个命令的作用是相同的。

```
$ hexo generate --deploy$ hexo deploy --generate
```

部署

Hexo 提供了快速方便的一键部署功能，让您只需一条命令就能将网站部署到服务器上。

```
$ hexo deploy
```

在开始之前，您必须先在 `_config.yml` 中修改参数，一个正确的部署配置中至少要有 `type` 参数，例如：

```
deploy:
  type: git
```

您可同时使用多个 deployer，Hexo 会依照顺序执行每个 deployer。

```
deploy:
- type: git
  repo:
- type: heroku
  repo:
```

Git

安装 hexo-deployer-git。

```
$ npm install hexo-deployer-git --save
```

修改配置。

```
deploy:
  type: git
  repo: <repository url>
  branch: [branch]
  message: [message]
```

参数	描述
`repo`	库（Repository）地址
`branch`	分支名称。如果您使用的是 GitHub 或 GitCafe 的话，程序会尝试自动检测。
`message`	自定义提交信息 (默认为 `Site updated: {{ now("YYYY-MM-DD HH:mm:ss") }}`)

Heroku

安装 hexo-deployer-heroku。

```
$ npm install hexo-deployer-heroku --save
```

修改配置。

```
deploy:
  type: heroku
  repo: <repository url>
  message: [message]
```

参数	描述
`repo`	Heroku 库（Repository）地址
`message`	自定提交信息 (默认为 `Site updated: {{ now("YYYY-MM-DD HH:mm:ss") }}`)

Rsync

安装 hexo-deployer-rsync。

```
$ npm install hexo-deployer-rsync --save
```

修改配置。

```
deploy:
  type: rsync
  host: <host>
  user: <user>
  root: <root>
  port: [port]
  delete: [true|false]
  verbose: [true|false]
  ignore_errors: [true|false]
```

参数	描述	默认值
`host`	远程主机的地址	
`user`	使用者名称	
`root`	远程主机的根目录	
`port`	端口	22
`delete`	删除远程主机上的旧文件	true
`verbose`	显示调试信息	true
`ignore_errors`	忽略错误	false

OpenShift

安装 hexo-deployer-openshift。

```
$ npm install hexo-deployer-openshift --save
```

修改配置。

```
deploy:
  type: openshift
  repo: <repository url>
  message: [message]
```

参数	描述
`repo`	OpenShift 库（Repository）地址
`message`	自定提交信息 (默认为 `Site updated: {{ now("YYYY-MM-DD HH:mm:ss") }}`)

FTPSync

安装 hexo-deployer-ftpsync。

```
$ npm install hexo-deployer-ftpsync --save
```

修改配置。

```
deploy:
  type: ftpsync
  host: <host>
  user: <user>
  pass: <password>
  remote: [remote]
  port: [port]
  ignore: [ignore]
  connections: [connections]
  verbose: [true|false]
```

参数	描述	默认值
`host`	远程主机的地址	
`user`	使用者名称	
`pass`	密码	
`remote`	远程主机的根目录	`/`
`port`	端口	21
`ignore`	忽略的文件或目录	
`connections`	使用的连接数	1
`verbose`	显示调试信息	false

其他方法

Hexo 生成的所有文件都放在 `public` 文件夹中，您可以将它们复制到您喜欢的地方。

自定义

永久链接（Permalinks）

您可以在 `_config.yml` 配置中调整网站的永久链接或者在每篇文章的 Front-matter 中指定。

变量

除了下列变量外，您还可使用 Front-matter 中的所有属性。

变量	描述
`:year`	文章的发表年份（4 位数）
`:month`	文章的发表月份（2 位数）
`:i_month`	文章的发表月份（去掉开头的零）
`:day`	文章的发表日期 (2 位数)
`:i_day`	文章的发表日期（去掉开头的零）
`:title`	文件名称
`:id`	文章 ID
`:category`	分类。如果文章没有分类，则是 `default_category` 配置信息。

您可在 `permalink_defaults` 参数下调整永久链接中各变量的默认值：

```
permalink_defaults:
  lang: en
```

示例

假设 `source/_posts` 文件夹中有个 `hello-world.md`，包含以下内容：

```
title: Hello World
date: 2013-07-14 17:01:34
categories:
- foo
- bar
```

参数	结果
`:year/:month/:day/:title/`	2013/07/14/hello-world
`:year-:month-:day-:title.html`	2013-07-14-hello-world.html
`:category/:title`	foo/bar/hello-world

多语种支持

若要建立一个多语种的网站，您可修改 `new_post_name` 和 `permalink` 参数，如下：

```
new_post_name: :lang/:title.md
permalink: :lang/:title/
```

当您建立新文章时，文章会被储存到：

```
$ hexo new "Hello World"
# => source/_posts/tw/Hello-World.md
```

而网址会是：

```
http://localhost:4000/tw/hello-world/
```

主题

创建 Hexo 主题非常容易，您只要在 `themes` 文件夹内，新增一个任意名称的文件夹，并修改 `_config.yml` 内的 `theme` 设定，即可切换主题。一个主题可能会有以下的结构：

```
.
├── _config.yml
├── languages
├── layout
├── scripts
└── source
```

_config.yml

主题的配置文件。修改时会自动更新，无需重启服务器。

languages

语言文件夹。请参见 国际化 (i18n)。

layout

布局文件夹。用于存放主题的模板文件，决定了网站内容的呈现方式，Hexo 内建 Swig 模板引擎，您可以另外安装插件来获得 EJS、Haml 或 Jade 支持，Hexo 根据模板文件的扩展名来决定所使用的模板引擎，例如：

```
layout.ejs   - 使用 EJS
layout.swig  - 使用 Swig
```

您可参考 模板 以获得更多信息。

scripts

脚本文件夹。在启动时，Hexo 会载入此文件夹内的 JavaScript 文件，请参见 插件 以获得更多信息。

source

资源文件夹，除了模板以外的 Asset，例如 CSS、JavaScript 文件等，都应该放在这个文件夹中。文件或文件夹开头名称为 `_` （下划线线）或隐藏的文件会被忽略。

如果文件可以被渲染的话，会经过解析然后储存到 `public` 文件夹，否则会直接拷贝到 `public` 文件夹。

发布

当您完成主题后，可以考虑将它发布到 主题列表，让更多人能够使用您的主题。在发布前建议先进行 主题单元测试，确保每一项功能都能正常使用。发布主题的步骤和 更新文档 非常类似。

1. Fork hexojs/site
2. 把库（repository）复制到电脑上，并安装所依赖的插件。

```
$ git clone https://github.com/&lt;username&gt;/site.git
$ cd site
$ npm install
```

3. 编辑 `source/_data/themes.yml` ，在文件中新增您的主题，例如：

```
- name: landscape
  description: A brand new default theme for Hexo.
  link: https://github.com/hexojs/hexo-theme-landscape
  preview: http://hexo.io/hexo-theme-landscape
  tags:
    - official
    - responsive
    - widget
    - two_column
    - one_column
```

4. 在 `source/themes/screenshots` 新增同名的截图档案，图片必须为 800x500 的 PNG 文件。

5. 推送（push）分支。

6. 建立一个新的合并申请（pull request）并描述改动。

模版

模板决定了网站内容的呈现方式，每个主题至少都应包含一个 `index` 模板，以下是各页面相对应的模板名称：

模板	用途	回调
`index`	首页	
`post`	文章	`index`
`page`	分页	`index`
`archive`	归档	`index`
`category`	分类归档	`archive`
`tag`	标签归档	`archive`

布局（Layout）

如果页面结构类似，例如两个模板都有页首（Header）和页脚（Footer），您可考虑通过「布局」让两个模板共享相同的结构。一个布局文件必须要能显示 `body` 变量的内容，如此一来模板的内容才会被显示，举例来说：

```
index
```

```
<!DOCTYPE html>
<html  <body<%-body</body</html
```

生成：

```
<!DOCTYPE html>
<html  <bodyindex</body</html
```

每个模板都默认使用 `layout` 布局，您可在 front-matter 指定其他布局，或是设为 `false` 来关闭布局功能，您甚至可在布局中再使用其他布局来建立嵌套布局。

局部模版（Partial）

局部模板让您在不同模板之间共享相同的组件，例如页首（Header）、页脚（Footer）或侧边栏（Sidebar）等，可利用局部模板功能分割为个别文件，让维护更加便利。举例来说：

```
<h1id"logo"<%=config.title</h1
```

```
<%-partialpartialheader<divid"content"Home page</div
```

生成：

```
<h1id"logo"My Site</h1<divid"content"Home page</div
```

局部变量

您可以在局部模板中指定局部变量并使用。

```
<h1id"logo"<%=title</h1
```

```
<%-partialpartialheadertitle:HelloWorld<divid"content"Home page</di
```

生成：

```
<h1id"logo"Hello World</h1<divid"content"Home page</div
```

优化

如果您的主题太过于复杂，或是需要生成的文件量太过于庞大，可能会大幅降低性能，除了简化主题外，您可以考虑 Hexo 2.7 新增的局部缓存（Fragment Caching） 功能。

本功能借鉴于 Ruby on Rails，它储存局部内容，下次便能直接使用缓存内容，可以减少文件夹查询并使生成速度更快。

它可用于页首、页脚、侧边栏等文件不常变动的位置，举例来说：

```
<%- fragment_cache('header'function{  return'<header></header>'
});
```

如果您使用局部模板的话，可以更简单：

```
<%- partial('header'true
```

但是，如果您开启了 `relative_link` 参数的话，请勿使用局部缓存功能，因为相对链接在每个页面可能不同。

变量

全局变量

变量	描述
`site`	网站变量
`page`	针对该页面的内容以及 front-matter 所设定的变量。
`config`	网站配置
`theme`	主题配置。继承自网站配置。
`_` (单下划线)	Lodash 函数库
`path`	当前页面的路径（不含根路径）
`url`	当前页面的完整网址
`env`	环境变量

网站变量

变量	描述
`site.posts`	所有文章
`site.pages`	所有分页
`site.categories`	所有分类
`site.tags`	所有标签

页面变量

页面（page）

变量	描述
`page.title`	页面标题
`page.date`	页面建立日期（Moment.js 对象）
`page.updated`	页面更新日期（Moment.js 对象）
`page.comments`	留言是否开启
`page.layout`	布局名称
`page.content`	页面的完整内容
`page.excerpt`	页面摘要
`page.more`	除了页面摘要的其余内容
`page.source`	页面原始路径
`page.full_source`	页面的完整原始路径
`page.path`	页面网址（不含根路径）。我们通常在主题中使用 `url_for(page.path)` 。
`page.permalink`	页面的完整网址
`page.prev`	上一个页面。如果此为第一个页面则为 `null` 。
`page.next`	下一个页面。如果此为最后一个页面则为 `null` 。
`page.raw`	文章的原始内容
`page.photos`	文章的照片（用于相簿）
`page.link`	文章的外部链接（用于链接文章）

文章 **(post):** 和 `page` 布局类似，但是添加了下列变量。

Variable	Description
`page.published`	如果该文章已发布则为True
`page.categories`	该文章的所有分类
`page.tags`	该文章的所有标签

首页（**index**）

变量	描述
`page.per_page`	每页显示的文章数量
`page.total`	总文章数
`page.current`	目前页数
`page.current_url`	目前分页的网址
`page.posts`	本页文章
`page.prev`	上一页的页数。如果此页是第一页的话则为 `0` 。
`page.prev_link`	上一页的网址。如果此页是第一页的话则为 `''` 。
`page.next`	下一页的页数。如果此页是最后一页的话则为 `0` 。
`page.next_link`	下一页的网址。如果此页是最后一页的话则为 `''` 。
`page.path`	当前页面的路径（不含根目录）。我们通常在主题中使用 `url_for(page.path)` 。

归档 **(archive)**：与 `index` 布局相同，但新增以下变量。

变量	描述
`page.archive`	等于 `true`
`page.year`	年份归档 (4位)
`page.month`	月份归档 (没有前导零的2位数)

分类 **(category)**：与 `index` 布局相同，但新增以下变量。

变量	描述
`page.category`	分类名称

标签 **(tag)**：与 `index` 布局相同，但新增以下变量。

变量	描述
`page.tag`	标签名称

辅助函数（Helpers）

辅助函数帮助您在模版中快速插入内容。辅助函数不能在源文件中使用。

网址

url_for

在路径前加上根路径，从 Hexo 2.7 开始您应该使用此函数而不是 `config.root + path`。

```
<%- url_for(path) %>
```

relative_url

取得与 `from` 相对的 `to` 路径。

```
<%- relative_url(from, to) %>
```

gravatar

插入 Gravatar 图片。 如果你不指定 `options` 参数，将会应用默认参数。否则，你可以将其设置为一个数字，这个数字将会作为 Gravatar 的大小参数。最后，如果你设置它一个对象，它将会被转换为 Gravatar 的一个查询字符串参数。

```
<%- gravatar(email, [options]);
```

示例：

```
<%- gravatar('a@abc.com'
// http://www.gravatar.com/avatar/b9b00e66c6b8a70f88c73cb6bdb06787
<%- gravatar('a@abc.com'40
// http://www.gravatar.com/avatar/b9b00e66c6b8a70f88c73cb6bdb06787
<%- gravatar('a@abc.com'40'http://example.com/image.png'
// http://www.gravatar.com/avatar/b9b00e66c6b8a70f88c73cb6bdb06787
```

HTML 标签

css

载入 CSS 文件。 `path` 可以是数组或字符串，如果 `path` 开头不是 `/` 或任何协议，则会自动加上根路径；如果后面没有加上 `.css` 扩展名的话，也会自动加上。

```
<%- css(path, ...) %>
```

示例：

```
<%- css('style.css'
// <link rel="stylesheet" href="/style.css" type="text/css">
<%- css(['style.css''screen.css'
// <link rel="stylesheet" href="/style.css" type="text/css">
// <link rel="stylesheet" href="/screen.css" type="text/css">
```

js

载入 JavaScript 文件。 `path` 可以是数组或字符串，如果 `path` 开头不是 `/` 或任何协议，则会自动加上根路径；如果后面没有加上 `.js` 扩展名的话，也会自动加上。

```
<%- js(path, ...) %>
```

示例：

```
<%- js('script.js'
// <script type="text/javascript" src="/script.js"></script>
<%- js(['script.js''gallery.js'
// <script type="text/javascript" src="/script.js"></script>
// <script type="text/javascript" src="/gallery.js"></script>
```

link_to

插入链接。

```
<%- link_to(path, [text], [options]) %>
```

参数	描述	默认值
`external`	在新视窗打开链接	false
`class`	Class 名称	
`id`	ID	

示例：

```
<%- link_to('http://www.google.com'
// <a href="http://www.google.com" title="http://www.google.com">ht
<%- link_to('http://www.google.com''Google'
// <a href="http://www.google.com" title="Google">Google</a>
<%- link_to('http://www.google.com''Google'true
// <a href="http://www.google.com" title="Google" target="_blank" r
```

mail_to

插入电子邮箱链接。

```
<%- mail_to(path, [text], [options]) %>
```

参数	描述
`class`	Class 名称
`id`	ID
`subject`	邮件主题
`cc`	抄送（CC）
`bcc`	密送（BCC）
`body`	邮件内容

示例：

```
<%- mail_to('a@abc.com'
// <a href="mailto:a@abc.com" title="a@abc.com">a@abc.com</a>
<%- mail_to('a@abc.com''Email'
// <a href="mailto:a@abc.com" title="Email">Email</a>
```

image_tag

插入图片。

```
<%- image_tag(path, [options]) %>
```

参数	描述
`alt`	图片的替代文字
`class`	Class 名称
`id`	ID
`width`	图片宽度
`height`	图片高度

favicon_tag

插入 favicon。

```
<%- favicon_tag(path) %>
```

feed_tag

插入 feed 链接。

```
<%- feed_tag(path, [options]) %>
```

参数	描述	默认值
`title`	Feed 标题	
`type`	Feed 类型	atom

条件函数

is_current

检查 `path` 是否符合目前页面的网址。开启 `strict` 选项启用严格比对。

```
<%- is_current(path, [strict]) %>
```

is_home

检查目前是否为首页。

```
<%- is_home() %>
```

is_post

检查目前是否为文章。

```
<%- is_post() %>
```

is_archive

检查目前是否为存档页面。

```
<%- is_archive() %>
```

is_year

检查目前是否为年度归档页面。

```
<%- is_year() %>
```

is_month

检查目前是否为月度归档页面。

```
<%- is_month() %>
```

is_category

检查目前是否为分类归档页面。 如果给定一个字符串作为参数，将会检查目前是否为指定分类。

```
<%- is_category() %><%-is_categoryhobby
```

is_tag

检查目前是否为标签归档页面。 如果给定一个字符串作为参数，将会检查目前是否为指定标签。

```
<%- is_tag() %><%-is_taghobby
```

字符串处理

trim

清除字符串开头和结尾的空格。

```
<%- trim(string) %>
```

strip_html

清除字符串中的 HTML 标签。

```
<%- strip_html(string) %>
```

示例：

```
<%- strip_html('It's not <b>important</b> anymore!') %>// It's not
```

titlecase

把字符串转换为正确的 Title case。

```
<%- titlecase(string) %>
```

示例：

```
<%- titlecase('this is an apple') %># This is an Apple
```

markdown

使用 Markdown 解析字符串。

```
<%- markdown(str) %>
```

示例：

```
<%- markdown('make me **strong**'
// make me <strong>strong</strong>
```

render

解析字符串。

```
<%- render(str, engine, [options]) %>
```

word_wrap

使每行的字符串长度不超过 `length` 。 `length` 预设为 80。

```
<%- word_wrap(str, [length]) %>
```

示例：

```
<%- word_wrap('Once upon a time'8
// Once upon\n a time
```

truncate

移除超过 `length` 长度的字符串。

```
<%- truncate(text, length) %>
```

示例：

```
<%- truncate('Once upon a time in a world far far away'17
// Once upon a ti...
<%- truncate('Once upon a time in a world far far away'17' '
// Once upon a...
<%- truncate('And they found that many people were sleeping better
// And they f... (continued)
```

模板

partial

载入其他模板文件，您可在 `locals` 设定区域变量。

```
<%- partial(layout, [locals], [options]) %>
```

参数	描述	默认值
`cache`	缓存（使用 Fragment cache）	`false`
`only`	限制局部变量。在模板中只能使用 `locals` 中设定的变量。	`false`

fragment_cache

局部缓存。它储存局部内容，下次使用时就能直接使用缓存。

```
<%- fragment_cache(id, fn);
```

示例：

```
<%- fragment_cache('header'function{  return'<header></header>'
}) %>
```

日期与时间

date

插入格式化的日期。 `date` 可以是 UNIX 时间、ISO 字符串、Date 对象或 Moment.js 对象。 `format` 默认为 `date_format` 配置信息。

```
<%- date(date, [format]) %>
```

示例：

```
<%- date(Date
// 2013-01-01
<%- date(Date'YYYY/M/D'
// Jan 1 2013
```

date_xml

插入 XML 格式的日期。 `date` 可以是 UNIX 时间、ISO 字符串、Date 对象或 Moment.js 对象。

```
<%- date_xml(date) %>
```

示例：

```
<%- date_xml(Date
// 2013-01-01T00:00:00.000Z
```

time

插入格式化的时间。 `date` 可以是 UNIX 时间、ISO 字符串、Date 对象或 Moment.js 对象。 `format` 默认为 `time_format` 配置信息。

```
<%- time(date, [format]) %>
```

示例：

```
<%- time(Date
// 13:05:12
<%- time(Date'h:mm:ss a'
// 1:05:12 pm
```

full_date

插入格式化的日期和时间。 `date` 可以是 UNIX 时间、ISO 字符串、Date 对象或 Moment.js 对象。 `format` 默认为 `date_format + time_format` 。

```
<%- full_date(date, [format]) %>
```

示例：

```
<%- full_date(newDate
// Jan 1, 2013 0:00:00
<%- full_date(newDate'dddd, MMMM Do YYYY, h:mm:ss a'
// Tuesday, January 1st 2013, 12:00:00 am
```

moment

Moment.js 函数库。

列表

list_categories

插入分类列表。

```
<%- list_categories([options]) %>
```

参数	描述	默认值
`orderby`	分类排列方式	name
`order`	分类排列顺序。 `1` ， `asc` 升序； `-1` ， `desc` 降序。	1
`show_count`	显示每个分类的文章总数	true
`style`	分类列表的显示方式。使用 `list` 以无序列表（unordered list）方式显示。	list
`separator`	分类间的分隔符号。只有在 `style` 不是 `list` 时有用。	,
`depth`	要显示的分类层级。 `0` 显示所有层级的分类； `-1` 和 `0` 很类似，但是显示不分层级； `1` 只显示第一层的分类。	0
`class`	分类列表的 class 名称。	category
`transform`	改变分类名称显示方法的函数	

list_tags

插入标签列表。

```
<%- list_tags([options]) %>
```

选项	描述	预设值
`orderby`	标签排列方式	name
`order`	标签排列顺序。 `1` , `asc` 升序； `-1` , `desc` 降序。	1
`show_count`	显示每个标签的文章总数	true
`style`	标签列表的显示方式。使用 `list` 以无序列表（unordered list）方式显示。	list
`separator`	标签间的分隔符号。只有在 `style` 不是 `list` 时有用。	,
`class`	标签列表的 class 名称。	tag
`transform`	改变标签名称显示方法的函数	
`amount`	要显示的标签数量（0 = 无限制）	0

list_archives

插入归档列表。

```
<%- list_archives([options]) %>
```

参数	描述	默认值
`type`	类型。此设定可为 `yearly` 或 `monthly` 。	monthly
`order`	排列顺序。 `1` , `asc` 升序； `-1` , `desc` 降序。	1
`show_count`	显示每个归档的文章总数	true
`format`	日期格式	MMMM YYYY
`style`	归档列表的显示方式。使用 `list` 以无序列表（unordered list）方式显示。	list
`separator`	归档间的分隔符号。只有在 `style` 不是 `list` 时有用。	,
`class`	归档列表的 class 名称。	archive
`transform`	改变归档名称显示方法的函数	

list_posts

插入文章列表。

```
<%- list_posts([options]) %>
```

参数	描述	默认值
`orderby`	文章排列方式	date
`order`	文章排列顺序。 `1` , `asc` 升序； `-1` , `desc` 降序。	-1
`style`	文章列表的显示方式。使用 `list` 以无序列表（unordered list）方式显示。	list
`separator`	文章间的分隔符号。只有在 `style` 不是 `list` 时有用。	,
`class`	文章列表的 class 名称。	post
`amount`	要显示的文章数量（0 = 无限制）	6
`transform`	改变文章名称显示方法的函数	

tagcloud

插入标签云。

```
<%- tagcloud([tags], [options]) %>
```

参数	描述	默认值
`min_font`	最小字体尺寸	10
`max_font`	最大字体尺寸	20
`unit`	字体尺寸的单位	px
`amount`	标签总量	40
`orderby`	标签排列方式	name
`order`	标签排列顺序。 `1` , `sac` 升序； `-1` , `desc` 降序	1
`color`	使用颜色	false
`start_color`	开始的颜色。您可使用十六进位值（ `#b700ff` ），rgba（ `rgba(183, 0, 255, 1)` ），hsla（ `hsla(283, 100%, 50%, 1)` ）或 颜色关键字。此变量仅在 `color` 参数开启时才有用。	
`end_color`	结束的颜色。您可使用十六进位值（ `#b700ff` ），rgba（ `rgba(183, 0, 255, 1)` ），hsla（ `hsla(283, 100%, 50%, 1)` ）或 颜色关键字。此变量仅在 `color` 参数开启时才有用。	

其他

paginator

插入分页链接。

```
<%- paginator(options) %>
```

参数	描述	默认值
`base`	基础网址	/
`format`	网址格式	page/%d/
`total`	分页总数	1
`current`	目前页数	0
`prev_text`	上一页链接的文字。仅在 `prev_next` 设定开启时才有用。	Prev
`next_text`	下一页链接的文字。仅在 `prev_next` 设定开启时才有用。	Next
`space`	空白文字	&hellp;
`prev_next`	显示上一页和下一页的链接	true
`end_size`	显示于两侧的页数	1
`mid_size`	显示于中间的页数	2
`show_all`	显示所有页数。如果开启此参数的话， `end_size` 和 `mid_size` 就没用了。	false

search_form

插入 Google 搜索框。

```
<%- search_form(options) %>
```

参数	描述	默认值
`class`	表单的 class name	search-form
`text`	搜索提示文字	Search
`button`	显示搜索按钮。此参数可为布尔值（boolean）或字符串，当设定是字符串的时候，即为搜索按钮的文字。	false

number_format

格式化数字。

```
<%- number_format(number, [options]) %>
```

参数	描述	默认值
`precision`	数字精度。此选项可为 `false` 或非负整数。	false
`delimiter`	千位数分隔符号	,
`separator`	整数和小数之间的分隔符号	.

示例：

```
<%- number_format(12345.671
// 12,345.68
<%- number_format(12345.674
// 12,345.6700
<%- number_format(12345.670
// 12,345
<%- number_format(12345.67''
// 12345.67
<%- number_format(12345.67'/'
// 12,345/67
```

open_graph

插入 open graph 资源。

```
<%- open_graph([options]) %>
```

参数	描述	默认值
`title`	页面标题 (`og:title`)	`page.title`
`type`	页面类型 (`og:type`)	blog
`url`	页面网址 (`og:url`)	`url`
`image`	页面图片 (`og:image`)	内容中的图片
`site_name`	网站名称 (`og:site_name`)	`config.title`
`description`	页面描述 (`og:desription`)	内容摘要或前 200 字
`twitter_card`	Twitter 卡片类型 (`twitter:card`)	summary
`twitter_id`	Twitter ID (`twitter:creator`)	
`twitter_site`	Twitter 网站 (`twitter:site`)	
`google_plus`	Google+ 个人资料链接	
`fb_admins`	Facebook 管理者 ID	
`fb_app_id`	Facebook 应用程序 ID	

toc

解析内容中的标题标签 (h1~h6) 并插入目录。

```
<%- toc(str, [options]) %>
```

参数	描述	默认值
`class`	Class 名称	toc
`list_number`	显示编号	true

示例：

```
<%- toc(page.content) %>
```

国际化（i18n）

若要让您的网站以不同语言呈现，您可使用国际化（internationalization）功能。请先在 `_config.yml` 中调整 `language` 设定，这代表的是预设语言，您也可设定多个语言来调整预设语言的顺位。

```
language: zh-tw

language:
- zh-tw
- en
```

语言文件

语言文件可以使用 YAML 或 JSON 编写，并放在主题文件夹中的 `languages` 文件夹。您可以在语言文件中使用 printf 格式。

模板

在模板中，透过 `__` 或 `_p` 辅助函数，即可取得翻译后的字符串，前者用于一般使用；而后者用于复数字符串。例如：

```
index:
  title: Home
  add: Add
  video:
    zero: No videos
    one: One video
    other: %d videos
```

```
<%= __('index.title'
// Home
<%= _p('index.video'3
// 3 videos
```

路径

您可在 front-matter 中指定该页面的语言，也可在 `_config.yml` 中修改 `i18n_dir` 设定，让 Hexo 自动侦测。

```
i18n_dir: :lang
```

`i18n_dir` 的预设值是 `:lang`，也就是说 Hexo 会捕获网址中的第一段以检测语言，举例来说：

```
/index.html => en
/archives/index.html => en
/zh-tw/index.html => zh-tw
```

捕获到的字符串唯有在语言文件存在的情况下，才会被当作是语言，因此例二 `/archives/index.html` 中的 `archives` 就不被当成是语言。

插件

Hexo 有强大的插件系统，使您能轻松扩展功能而不用修改核心模块的源码。在 Hexo 中有两种形式的插件：

脚本（Scripts）

如果您的代码很简单，建议您编写脚本，您只需要把 JavaScript 文件放到 `scripts` 文件夹，在启动时就会自动载入。

插件（Packages）

如果您的代码较复杂，或是您想要发布到 NPM 上，建议您编写插件。首先，在 `node_modules` 文件夹中建立文件夹，文件夹名称开头必须为 `hexo-` ，如此一来 Hexo 才会在启动时载入否则 Hexo 将会忽略它。

文件夹内至少要包含 2 个文件：一个是主程序，另一个是 `package.json` ，描述插件的用途和所依赖的插件。

```
.
├── index.js
└── package.json
```

`package.json` 中至少要包含 `name` , `version` , `main` 属性，例如：

```
{  "name"hexo-my-plugin",  "version"0.0.1",  "main"index"
```

工具

您可以使用 Hexo 提供的官方工具插件来加速开发：

- hexo-fs：文件 IO
- hexo-util：工具程式
- hexo-i18n：本地化（i18n）
- hexo-pagination：生成分页资料

发布

当您完成插件后，可以考虑将它发布到 插件列表，让更多人能够使用您的插件。发布插件的步骤和 更新文件 非常类似。

1. Fork hexojs/site

2. 把库（repository）复制到电脑上，并安装所依赖的插件。

   ```
   $ git clone https://github.com/&lt;username&gt;/site.git
   $ cd site
   $ npm install
   ```

3. 编辑 `source/_data/plugins.yml`，在档案中新增您的插件，例如：

   ```
   - name: hexo-server
     description: Server module for Hexo.
     link: https://github.com/hexojs/hexo-server
     tags:
       - official
       - server
       - console
   ```

4. 推送（push）分支。
5. 建立一个新的合并申请（pull request）并描述改动。

其他

问题解答

在使用 Hexo 时，您可能会遇到一些问题，下列的常见问题解答可能会对您有所帮助。如果您在这里找不道解答，可以在 GitHub 或 Google Group 上提问。

YAML 解析错误

```
JS-YAML: incomplete explicit mapping pair; a key node is missed at
      last_updated: Last updated: %s
```

如果 YAML 字符串中包含冒号（ `:` ）的话，请加上引号。

```
JS-YAML: bad indentation of a mapping entry at line 18, column 31:
      last_updated:"Last updated: %s"
```

请确认您使用空格进行缩进（Soft tab），并确认冒号后有加上一个空格。

您可参阅 YAML 规范 以取得更多信息。

EMFILE 错误

```
Error: EMFILE, too many open files
```

虽然 Node.js 有非阻塞 I/O，同步 I/O 的数量仍被系统所限制，在生成大量静态文件的时候，您可能会碰到 EMFILE 错误，您可以尝试提高同步 I/O 的限制数量来解决此问题。

```
$ ulimit10000
```

Git 部署问题

```
fatal: 'username.github.io' does not appear to be a git repository
```

请确认您已经在电脑上 配置 git，或改用 HTTPS 库（repository）地址。

服务器问题

```
Error: listen EADDRINUSE
```

您可能同时开启两个 Hexo 服务器，或者有其他应用程序正在占用相同的端口，请尝试修改 `port` 参数，或是在启动 Hexo 服务器时加上 `-p` 选项。

```
$ hexo server -p 5000
```

插件安装问题

```
npm ERR! node-waf configure build
```

当您尝试安装以 C/C++ 或其他非 JavaScript 语言所编写的插件时，可能会遇到此类问题，请确认您已经在电脑上安装相对应的编译器。

DTrace 错误 （Mac OS X）

```
{ [Error: Cannot find module './build/Release/DTraceProviderBinding
{ [Error: Cannot find module './build/default/DTraceProviderBinding
{ [Error: Cannot find module './build/Debug/DTraceProviderBindings
```

DTrace 安装可能有错误，使用下列命令:

```
$ npm install hexo --no-optional
```

参考 #1326

在 Jade 或 Swig 遍历资料

Hexo 使用 Warehouse 存储资料，它不是一般数组所以必须先进行类型转型才能遍历。

```
{% for post in site.posts.toArray() %}
{% endfor %}
```

资料没有更新

有时资料可能没有被更新，或是生成的文件与修改前的相同，您可以尝试清除缓存并再执行一次。

```
$ hexo clean
```

泄露（Escape）内容

Hexo 使用 Nunjucks 来解析文章（旧版本使用 Swig，两者语法类似），内容若包含 `{{ }}` 或 `{% %}` 可能导致解析错误，您可以用 `raw` 标签包裹来避免潜在问题发生。

```
{% raw %}
Hello {{ sensitive }}
{% endraw %}
```

ENOSPC 错误 （Linux）

运行 `$ hexo server` 命令有时会返回这样的错误：

```
Error: watch ENOSPC ...
```

它可以用过运行 `$ npm dedupe` 来解决，如果不起作用的话，可以尝试在 Linux 终端中运行下列命令：

```
$ echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl
```

这将会提高你能监视的文件数量。

贡献

开发

我们非常欢迎您加入 Hexo 的开发，这份文件将帮助您了解开发流程。

开始之前

请遵守以下准则：

- 遵守 Google JavaScript 代码风格。
- 使用 2 个空格缩排。
- 不要把逗号放在最前面。

工作流程

1. Fork hexojs/hexo
2. 把库（repository）复制到电脑上，并安装所依赖的插件。

   ```
   $ git clone https://github.com/&lt;username&gt;/hexo.git
   $ cd hexo
   $ npm install
   $ git submodule update --init
   ```

3. 新增一个功能分支。

   ```
   $ git checkout -b new_feature
   ```

4. 开始开发。
5. 推送（push）分支。

   ```
   $ git push origin new_feature
   ```

6. 建立一个新的合并申请（pull request）并描述变动。

注意事项

- 不要修改 `package.json` 的版本号。
- 只有在测试通过的情况下您的合并申请才会被批准，在提交前别忘了进行测试。

```
$ npm test
```

更新文档

Hexo 文档开放源代码，您可以在 hexojs/site 找到源代码。

工作流程

1. Fork hexojs/site
2. 把库（repository）复制到电脑上，并安装所依赖的插件。

```
$ git clone https://github.com/&lt;username&gt;/site.git
$ cd site
$ npm install
```

3. 开始编辑文件，您可以通过服务器预览变动。

```
$ hexo server
```

4. 推送（push）分支。

5. 建立一个新的合并申请（pull request）并描述变动。

翻译

1. 在 `source` 资料夹中建立一个新的语言资料夹（全小写）。
2. 把 `source` 资料夹中相关的文件（Markdown 和模板文件）复制到新的语言资料夹中。
3. 在 `source/_data/language.yml` 中新增语言。
4. 将 `en.yml` 复制到 `themes/navy/languages` 中并命名为语言名称（全小写）。

反馈问题

当您在使用 Hexo 时遇到问题，您可以尝试在 问题解答 中寻找解答，或是在 GitHub 或 Google Group 上提问。如果你没有找答案，请在 Github 报告它。

1. 在 调试模式 中重现问题。
2. 运行 `hexo version` 并检查版本信息。
3. 把调试信息和版本信息都贴到 GitHub。

API

核心

概述

本文件提供您更丰富的 API 信息，使您更容易修改 Hexo 源代码或编写插件。如果您只是想查询 Hexo 的基本使用方法，请参阅 文档。

在开始之前，请注意本文件仅适用于 Hexo 3 及以上版本。

初始化

首先，我们必须建立一个 Hexo 实例（instance），第一个参数是网站的根目录，也就是 `base_dir` ，而第二个参数则是初始化的选项。接著执行 `init` 方法后，Hexo 会加载插件及配置文件。

```
varrequire'hexo'
varnew
hexo.init().then(function{  // ...
});
```

参数	描述	默认值
`debug`	开启调试模式。在终端中显示调试信息，并在根目录中存储 `debug.log` 日志文件。	`false`
`safe`	开启安全模式。不加载任何插件。	`false`
`silent`	开启安静模式。不在终端中显示任何信息。	`false`
`config`	指定配置文件的路径。	`_config.yml`

载入文件

Hexo 提供了两种方法来载入文件： `load` , `watch` ，前者用于载入 `source` 文件夹内的所有文件及主题资源；而后者除了执行 `load` 以外，还会继续监视文件变动。

这两个方法实际上所做的，就是载入文件列表，并把文件传给相对应的处理器（Processor），当文件全部处理完毕后，就执行生成器（Generator）来建立路由。

```
hexo.load().then(function{  // ...
});hexo.watch().then(function{  // 之后可以调用 hexo.unwatch()，停止监
});
```

执行指令

您可以通过 `call` 方法来调用控制台（Console），第一个参数是控制台的名称，而第二个参数是选项——根据不同控制台有所不同。

```
hexo.call('generate'function{  // ...
});
```

结束

当指令完毕后，请执行 `exit` 方法让 Hexo 退出结束前的准备工作（如存储资料库）。

```
hexo.call('generate'function{  return
}).catch(functionerr{  return
});
```

事件

Hexo 继承了 EventEmitter，您可以用 `on` 方法监听 Hexo 所发布的事件，也可以使用 `emit` 方法对 Hexo 发布事件，更详细的说明请参阅 Node.js 的 API。

deployBefore

在部署完成前发布。

deployAfter

在部署成功后发布。

exit

在 Hexo 结束前发布。

generateBefore

在静态文件生成前发布。

generateAfter

在静态文件生成后发布。

new

在文章文件建立后发布。该事件返回文章参数。

```
hexo.on('new'functionpost{  //
});
```

资料	描述
`post.path`	文章文件的完整路径
`post.content`	文章文件的内容

processBefore

在处理原始文件前发布。此事件会返回一个地址，代表 Box（Box）的根目录。

processAfter

在原始文件处理后发布。此事件会返回一个地址，代表 Box（Box）的根目录。

ready

在初始化完成后发布。

局部变量

局部变量用于模版渲染，也就是模版中的 `site` 变量。

默认变量

变量	描述
`posts`	所有文章
`pages`	所有分页
`categories`	所有分类
`tags`	所有标签

获取变量

```
hexo.locals.get('posts'
```

设置变量

```
hexo.locals.set('posts'function{   return
});
```

移除变量

```
hexo.locals.remove('posts'
```

获取所有变量

```
hexo.locals.toObject();
```

清除缓存

```
hexo.locals.invalidate();
```

路由

路由存储了网站中所用到的所有路径。

获取路径

`get` 方法会传回一个 Stream，例如把该路径的资料存储到某个指定位置。

```
var'index.html'
var'somewhere'
data.pipe(dest);
```

设置路径

您可以在 `set` 方法中使用字符串、Buffer 或函数，如下：

```
// String
hexo.route.set('index.html''index'
// Buffer
hexo.route.set('index.html'new'index'
// Function (Promise)
hexo.route.set('index.html'function{  returnnewPromisefunctionresol
  });});// Function (Callback)
hexo.route.set('index.html'functioncallback{  callback(null'index'
});
```

您还可以设置该路径是否更新，这样在生成文件时便能忽略未更动的文件，加快生成时间。

```
hexo.route.set('index.html'
    data: 'index'
    modified: false
});// hexo.route.isModified('index.html') => false
```

移除路径

```
hexo.route.remove('index.html'
```

获得路由表

```
hexo.route.list();
```

格式化路径

`format` 方法可将字符串转为合法的路径。

```
hexo.route.format('archives/'
// archives/index.html
```

Box

「Box」是 Hexo 用来处理特定文件夹中的文件的容器，在 Hexo 中有两个 Box，分别是 `hexo.source` 和 `hexo.theme`，前者用于处理 `source` 文件夹，而后者用于处理主题文件夹。

载入文件

Box 提供了两种方法来载入文件：`process`，`watch`，前者用于载入文件夹内的所有文件；而后者除了执行 `process` 以外，还会继续监视文件变动。

```
box.process().then(function{  // ...
});box.watch().then(function{  // 之后可调用 box.unwatch()，停止监视文
});
```

比对路径

Box 提供了多种比对路径的模式，您可以以使用正则表达式（regular expression）、函数、或是一种类似于 Express 的路径字符串，例如：

```
posts/:id => posts/89
posts/*path => posts/2015/title
```

您可以以参考 util.Pattern 以获得更多信息。

处理器（Processor）

处理器（Processor）是 Box 中非常重要的元素，它用于处理文件，您可以使用上述的路径对比来限制该处理器所要处理的文件类型。使用 `addProcessor` 来添加处理器。

```
box.addProcessor('posts/:id'functionfile{  //
});
```

Box 在处理时会把目前处理的文件内容（ `file` ）传给处理器，您可以通过此参数获得该文件的数据。

属性	描述
`source`	文件完整路径
`path`	文件相对于 Box 的路径
`type`	文件类型。有 `create` , `update` , `skip` , `delete` 。
`params`	从路径对比中取得的信息

Box 还提供了一些方法，让您无须手动处理文件 I/O。

方法	描述
`read`	读取文件
`readSync`	同步读取文件
`stat`	读取文件状态
`statSync`	同步读取文件状态
`render`	渲染文件
`renderSync`	同步渲染文件

渲染

在 Hexo 中，有两个方法可用于渲染文件或字符串，分别是非同步的 `hexo.render.render` 和同步的 `hexo.render.renderSync` ，这两个方法的使用方式十分类似，因此以下仅以非同步的 `hexo.render.render` 为例。

渲染字符串

在渲染字符串时，您必须指定 `engine` ，如此一来 Hexo 才知道该使用哪个渲染引擎来渲染。

```
hexo.render.render({text: 'example''swig'functionresult{  // ...
});
```

渲染文件

在渲染文件时，您无须指定 `engine` ，Hexo 会自动根据扩展名猜测所要使用的渲染引擎，当然您也可以使用 `engine` 指定。

```
hexo.render.render({path: 'path/to/file.swig'functionresult{  // .
});
```

渲染选项

在渲染时，您可以向第二个参数中传入参数。

```
hexo.render.render({text: '''foo'functionresult{  // ...
});
```

after_render 过滤器

在渲染完成后，Hexo 会自动执行相对应的 `after_render` 过滤器，举例来说，我们可以通过这个功能实现 JavaScript 的压缩。

```
varrequire'uglify-js'
hexo.extend.filter.register('after_render:js'functionstr, data{  va
  return
});
```

检查文件是否可被渲染

您可以通过 `isRenderable` 或 `isRenderableSync` 两个方法检查文件路径是否可以被渲染，只有在相对应的渲染器（renderer）已注册的情况下才会返回 true。

```
hexo.render.isRenderable('layout.swig'// true
hexo.render.isRenderable('image.png'// false
```

获取文件的输出扩展名

您可以通过 `getOutput` 方法取得文件路径输出后的扩展名，如果文件无法渲染，则会返回空字符串。

```
hexo.render.getOutput('layout.swig'// html
hexo.render.getOutput('image.png'// '''
```

文章

新建文章

```
hexo.post.create(data, replace);
```

参数	描述
`data`	数据
`replace`	替换现有文件

您可以在资料中指定文章的属性，除了以下属性之外，其他属性也会被加到 front-matter 中。

属性	描述
`title`	标题
`slug`	网址
`layout`	布局。默认为 `default_layout` 参数。
`path`	路径。默认会根据 `new_post_path` 参数创建文章路径。
`date`	日期。默认为当前时间。

发布草稿

```
hexo.post.publish(data, replace);
```

参数	描述
`data`	资料
`replace`	替换现有文件

您可以在资料中指定文章的属性，除了以下的属性之外，其他属性也会被加到 front-matter 中。

属性	描述
`slug`	文件名称（必须）
`layout`	布局。默认为 `default_layout` 参数。

渲染

```
hexo.post.render(source, data);
```

参数	描述
`source`	文件的完整路径（可忽略）
`data`	数据

资料中必须包含 `content` 属性，如果没有的话，会尝试读取原始文件。此函数的执行顺序为：

- 执行 `before_post_render` 过滤器
- 使用 Markdown 或其他渲染器渲染（根据扩展名而定）
- 使用 Nunjucks 渲染
- 执行 `after_post_render` 过滤器

脚手架（Scaffold）

获得脚手架

```
hexo.scaffold.get(name);
```

设置脚手架

```
hexo.scaffold.set(name, content);
```

移除脚手架

```
hexo.scaffold.remove(name);
```

主题

`hexo.theme` 除了继承 Box 外，还具有存储模板的功能。

获取模板

```
hexo.theme.getView(path);
```

设置模板

```
hexo.theme.setView(path, data);
```

移除模板

```
hexo.theme.removeView(path);
```

模板

模板本身有两个方法可供使用： `render` 和 `renderSync` 。两者功能一样，只是前者为非同步函数，而后者为同步函数，因此仅以 `render` 演示调用方法。

```
var'layout.swig'
view.render({foo: 12functionresult{  // ...
});
```

您可以以向 `render` 方法传入对象作为参数， `render` 方法会先使用对应的渲染引擎进行解析，并加载 辅助函数。渲染完成后，会检测布局（layout）是否存在，当 `layout` 设为 `false` 或不存在时，就会直接返回渲染结果。

扩展

控制台（Console）

控制台是 Hexo 与开发者之间沟通的桥梁。

概要

```
hexo.extend.console.register(name, desc, options, functionargs{  //
});
```

参数	描述
`name`	名称
`desc`	描述
`options`	选项

在函数中会传入 `args` 参数，此参数是使用者在终端中所传入的参数，是一个经 Minimist 解析的对象。

选项

用法

控制台的操作方法，例如：

```
{usage: '[layout] <title>'
// hexo new [layout] <title>
```

参数

控制台各个参数的说明，例如：

```
{  arguments
    {name: 'layout''Post layout'
    {name: 'title''Post title'
  ]}
```

选项

控制台的选项，例如：

```
{  options: [    {name: '-r, --replace''Replace existing files'
  ]}
```

描述

控制台更详细的说明。

范例

```
hexo.extend.console.register('config''Display configuration'functic
});
```

部署器（Deployer）

部署器帮助开发者将网站快速部署到远程服务器上，避免了复杂的指令。

概要

```
hexo.extend.deployer.register(name, functionargs{  // ...
});
```

在函数中会传入 `args` 参数，该参数包含了 `_config.yml` 中的 `deploy` 参数值，以及开发者在终端中所传入的参数。

过滤器（Filter）

过滤器用于修改特定文件，Hexo 将这些文件依序传给过滤器，而过滤器可以针对文件进行修改，这个概念借鉴自 WordPress。

概要

```
hexo.extend.filter.register(type, function{}, priority);
```

您可以指定过滤器的优先级 `priority` ， `priority` 值越低，过滤器会越早执行，默认的 `priority` 是 10。

执行过滤器

```
hexo.extend.filter.exec(type, data, options);hexo.extend.filter.exe
```

选项	描述
`context`	Context
`args`	参数。必须为数组。

`data` 会作为第一个参数传入每个过滤器，而您可以在过滤器中通过返回值改变下一个过滤器中的 `data` ，如果什么都没有返回的话则会保持原本的 `data` 。您还可以使用 `args` 指定过滤器的其他参数。举例来说：

```
hexo.extend.filter.register('test'functiondata, arg1, arg2{  // dat
  // arg1 === 'foo'
  // arg2 === 'bar'
    return'something'
});hexo.extend.filter.register('test'functiondata, arg1, arg2{  //
});hexo.extend.filter.exec('test''some data'
  args: ['foo''bar'
});
```

您也可以使用以下方法来执行过滤器：

```
hexo.execFilter(type, data, options);hexo.execFilterSync(type, data
```

移除过滤器

```
hexo.extend.filter.unregister(type, filter);
```

过滤器列表

以下是 Hexo 所使用的过滤器。

before_post_render

在文章开始渲染前执行。您可以参考 文章渲染 以了解执行顺序。

举例来说，把标题转为小写：

```
hexo.extend.filter.register('before_post_render'functiondata{  data
});
```

after_post_render

在文章渲染完成后执行。您可以参考 文章渲染 以了解执行顺序。

举例来说，把 `@username` 取代为 Twitter 的开发者链接。

```
hexo.extend.filter.register('after_post_render'functiondata{  data
  return
});
```

before_exit

在 Hexo 即将结束时执行，也就是在 `hexo.exit` 被调用后执行。

```
hexo.extend.filter.register('before_exit'function{  // ...
});
```

before_generate

在生成器解析前执行。

```
hexo.extend.filter.register('before_generate'function{  // ...
});
```

after_generate

在生成器解析后执行。

```
hexo.extend.filter.register('after_generate'function{  // ...
});
```

template_locals

修改模板的 局部变量。

举例来说，在模板的局部变量中新增当前时间：

```
hexo.extend.filter.register('template_locals'functionlocals{  local
  return
});
```

after_init

在 Hexo 初始化完成后执行，也就是在 `hexo.init` 执行完成后执行。

```
hexo.extend.filter.register('after_init'function{  // ...
});
```

new_post_path

用来决定新建文章的路径，在建立文章时执行。

```
hexo.extend.filter.register('new_post_path'functiondata, replace{
});
```

post_permalink

用来决定文章的永久链接。

```
hexo.extend.filter.register('post_permalink'functiondata{  // ...
});
```

after_render

在渲染后执行，您可以参考 渲染 以了解更多信息。

server_middleware

新增服务器的 Middleware。 `app` 是一个 Connect 实例。

举例来说，在响应头中新增 `X-Powered-By: Hexo` 。

```
hexo.extend.filter.register('server_middleware'functionapp{  app.us
    next();  });});
```

生成器（Generator）

生成器根据处理后的原始文件建立路由。

概要

```
hexo.extend.generator.register(name, functionlocals{});
```

在函数中会传入一个 `locals` 参数，等同于 网站变量，请尽量利用此参数取得网站数据，避免直接存取资料库。

更新路由

```
hexo.extend.generator.register('test'functionlocals{  // Object
  return
    path: 'foo'
    data: 'foo'
  };    // Array
  return
    {path: 'foo''foo'
    {path: 'bar''bar'
  ];});
```

属性	描述
`path`	路径。不可包含开头的 `/` 。
`data`	数据
`layout`	布局。指定用于渲染的模板，可为字符串或数组，如果省略此属性的话则会直接输出 `data` 。

在原始文件更新时，Hexo 会执行所有生成器并重建路由，请直接回传资料，不要直接操作路由。

范例

归档页面

在 `archives/index.html` 建立一归档页面，把所有文章当作资料传入模板内，这个资料也就等同于模板中的 `page` 变量。

然后，设置 `layout` 属性好让 Hexo 使用主题模板来渲染，在此例中同时设定了两个布局，当 `archive` 布局不存在时，会继续尝试 `index` 布局。

```
hexo.extend.generator.register('archive'functionlocals{  return
    path: 'archives/index.html'
    data: locals.posts,    layout: ['archive''index'
  }});
```

有分页的归档页面

您可以通过 hexo-pagination 这个方便的官方工具来轻松建立分页归档。

```
varrequire'hexo-pagination'
hexo.extend.generator.register('archive'functionlocals{  return'arc
    perPage: 10
    layout: ['archive''index'
    data: {}  });});
```

生成所有文章

遍历 `locals.posts` 中的所有文章并生成所有文章的路由。

```
hexo.extend.generator.register('post'functionlocals{  returnfunctio
      path: post.path,      data: post,      layout: 'post'
    };  });});
```

复制文件

这次不直接在 `data` 中返回数据而是返回一个函数，如此一来这个路由唯有在使用时才会建立 `fs.ReadStream` 。

```
varrequire'hexo-fs'
hexo.extend.generator.register('asset'functionlocals{  return
    path: 'file.txt'
    data: function{      return'path/to/file.txt'
    }  };});
```

辅助函数（Helper）

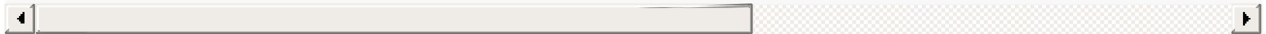
辅助函数帮助您在模板中快速插入内容，建议您把复杂的代码放在辅助函数而非模板中。

概要

```
hexo.extend.helper.register(name, function{});
```

范例

```
hexo.extend.helper.register('js'functionpath{  return'<script type=
});
```



```
<%- js('script.js'
// <script type="text/javascript" src="script.js"></script>
```

迁移器（Migrator）

迁移器帮助开发者从其他系统迁移到 Hexo。

概要

```
hexo.extend.migrator.register(name, functionargs{  // ...
});
```

在函数中需要传入 `args` 参数，该参数包含了开发者在终端中所传入的参数。

处理器（Processor）

处理器用于处理 `source` 文件夹内的原始文件。

概要

```
hexo.extend.processor.register(rule, functionfile{});
```

完整说明请参考 Box。

渲染引擎（Renderer）

渲染引擎用于渲染内容。

概要

```
hexo.extend.renderer.register(name, output, functiondata, options{]
```

参数	描述
`name`	输入的扩展名（小写，不含开头的 `.`）
`output`	输出的扩展名（小写，不含开头的 `.`）
`sync`	同步模式

渲染函数中会传入两个参数：

参数	描述
`data`	包含两个属性：文件路径 `path` 和文件内容 `text`。 `path` 不一定存在。
`option`	选项

范例

非同步模式

```
varrequire'stylus'
// Callback
hexo.extend.renderer.register('styl''css'functiondata, options, cal
});// Promise
hexo.extend.renderer.register('styl''css'functiondata, options{  re
  });});
```

同步模式

```
varrequire'ejs'
hexo.extend.renderer.register('ejs''html'functiondata, options{  op
}, true
```

标签插件（Tag）

标签插件帮助开发者在文章中快速插入内容。

概要

```
hexo.extend.tag.register(name, functionargs, content{}, options);
```

标签函数会传入两个参数： `args` 和 `content` ，前者代表开发者在使用标签插件时传入的参数，而后者则是标签插件所覆盖的内容。

从 Hexo 3 开始，因为新增了非同步渲染功能，而改用 Nunjucks 作为渲染引擎，其行为可能会与过去使用的 Swig 有些许差异。

选项

ends

使用结束标签，此选项默认为 `false` 。

async

开启非同步模式，此选项默认为 `false` 。

范例

没有结束标签

插入 Youtube 影片。

```
hexo.extend.tag.register('youtube'functionargs{   var0
  return'<div class="video-container"><iframe width="560" height="3
});
```

有结束标签

插入 pull quote。

```
hexo.extend.tag.register('pullquote'functionargs, content{  var' '
  return'<blockquote class="pullquote''">''</blockquote>'
}, {ends: true
```

非同步渲染

插入文件。

```
varrequire'hexo-fs'
varrequire'path'
hexo.extend.tag.register('include_code'functionargs{  var0
  var
    returnfunctioncontent{    return'<pre><code>''</code></pre>'
  });}, {asynctrue
```